public abstract class Figura

{ protected Color color;

protected String tipo;

/\*\* Constructor por defecto:

crea una figura por defecto de color negro (Color.black) y tipo "indefinido"

\*/

public Figura() {

color = Color.black;

tipo = "indefinido";

}

/\*\* Constructor que permite crear una figura de un color y un tipo determinados

\* @param color Color de la figura

\* @param tipo Descripcion del tipo de figura

\*/

public Figura(Color color, String tipo) {

this.color = color;

this.tipo = tipo;

}

public String getTipo() { return tipo; }

public Color getColor() { return color; }

public void setColor(Color color){ this.color = color; }

public String toString(){

return "Figura de tipo " + tipo + " y color " + color;

}

/\*\* Devuelve el true si la figura actual es igual a la que recibe como parametro.

\* Se considera que dos figuras son iguales si tienen el mismo tipo, color y area

\* @param x Figura a comparar con la actual

\* @return True si ambas figuras son iguales y false en caso contrario

\*/

public boolean equals(Object x) {

Figura f = (Figura) x;

return (color.equals(f.color) && tipo.equals(f.tipo) && this.area() == f.area());

}

/\*\* Devuelve el area de la figura

\* @return Valor real con el area de la figura

\*/

public abstract double area();

/\*\* Dibuja la figura actual sobre una superficie de dibujo

\* @param x Coordenada horizontal para el centro de la figura

\* @param y Coordenada vertical para el centro de la figura

\* @param g Superficie de dibujo de tipo java.awt.Graphics

\*/

public abstract void dibujar(int x, int y, Graphics g);

}

public class Circulo extends Figura

{ private double radio;

/\*\* Constructor por defecto: círculo de color negro (Color.black) y radio 10.0

\*/

public Circulo() {

super(Color.black, "c\u00edrculo");

radio = 10.0;

}

/\*\* Constructor que permite crear un circulo de un color y un radio determinados

\* @param color Color del circulo

\* @param radio Radio del circulo

\*/

public Circulo(Color color, double radio) {

super(color, "c\u00edrculo");

this.radio = radio;

}

public double getRadio() { return this.radio; }

/\*\* Devuelve el area del circulo

\* @return Valor real con el area del circulo

\*/

public double area() { return Math.PI \* radio \* radio; }

/\*\* Devuelve una descripción del círculo

\* @return Cadena de texto con la descripción del círculo

\*/

public String toString(){

return "C\u00edrculo de color " + color + " y radio " +

String.format(new java.util.Locale("US"),"%.2f", this.radio);

}

public void dibujar(int x, int y, Graphics g) {

int diametro = (int) (2 \* radio);

g.setColor(color);

g.fillOval(x - (int) radio, y - (int) radio, diametro, diametro);

}

}

public class Rectangulo extends Figura

{ protected double base;

protected double altura;

/\*\* Constructor por defecto: rectangulo de color (Color.black), base 10.0 y altura 10.0

\*/

public Rectangulo() {

super(Color.black, "rect\u00e1ngulo");

base = 10.0;

altura = 10.0;

}

/\*\* Constructor rectangulo de un color, una base y una altura determinados

\* @param color Color del rectangulo

\* @param base Base del rectangulo

\* @param altura Altura del rectangulo

\*/

public Rectangulo(Color color, double base, double altura) {

super(color, "rect\u00e1ngulo");

this.base = base;

this.altura = altura;

}

/\*\* Consultor que devuelve la base del rectangulo

\* @return Base del rectangulo

\*/

public double getBase() { return base; }

/\*\* Consultor que devuelve la altura del rectangulo

\* @return Altura del rectangulo

\*/

public double getAltura() { return altura; }

/\*\* Modificador para la base del rectangulo

\* @param base Nueva base del rectangulo

\*/

public void setBase(double base) { this.base = base; }

/\*\* Modificador para la altura del rectangulo

\* @param altura Nueva altura del rectangulo

\*/

public void setAltura(double altura) { this.altura = altura; }

/\*\* Devuelve el area del rectangulo

\* @return Valor real con el area del rectangulo

\*/

public double area(){ return base \* altura; }

/\*\* Devuelve una descripcion del rectangulo

\* @return Cadena de texto con la descripcion del rectangulo

\*/

public String toString() {

return "Rect\u00e1ngulo " + color + " de base " + base + " y altura " + altura;

}

public void dibujar(int x, int y, Graphics g) {

g.setColor(color);

g.fillRect(x - (int) (base / 2), y - (int) (altura / 2), (int) base, (int) altura);

}

}

public class Cuadrado extends Rectangulo

{ /\*\* Constructor por defecto: cuadrado de color (Color.black) y lado 10.0

\*/

public Cuadrado()

{ super();

tipo = "cuadrado";

}

/\*\* Constructor que permite crear un cuadrado de un color y un lado determinados

\* @param color Color del cuadrado

\* @param lado Lado del cuadrado

\*/

public Cuadrado(Color color, double lado) {

super(color, lado, lado);

tipo = "cuadrado";

}

/\*\* Consultor que devuelve el lado del cuadrado

\* @return Lado del cuadrado

\*/

public double getLado() { return base; }

/\*\* Modificador para el lado del cuadrado

\* @param radio Nuevo lado del cuadrado

\*/

public void setLado(double lado) { base = altura = lado; }

/\*\* Modificador para la base del cuadrado. Este metodo se sobreescribe

\* pues al modificar la base se debe modificarse la altura para coincidir

\* @param base Nueva base (y altura) del cuadrado

\*/

public void setBase(double base) { this.base = this.altura = base; }

/\*\* Modificador para la altura del cuadrado. Este método se sobreescribe

\* al modificar la altura se debe modificarse la base para coincidir

\* @param altura Nueva altura (y base) del cuadrado

\*/

public void setAltura(double altura) { this.base = this.altura = altura; }

/\*\* Devuelve una descripcion del cuadrado

\* @return Cadena de texto con la descripcion del cuadrado

\*/

public String toString(){

return "Cuadrado " + color + " de lado " + getLado();

}

}

public class ArrayGrupoDeFiguras

{

private int talla;

private Figura elArray[];

private static final int CAPACIDAD\_DEL\_ARRAY = 10;

/\*\* Constructor: crea un grupo de circulos vacio

\*/

public ArrayGrupoDeFiguras() {

elArray = new Figura[CAPACIDAD\_DEL\_ARRAY];

talla = 0;

}

/\*\* Inserta una figura en el grupo

\* @param c Figura a insertar

\*/

public void insertar(Figura c) {

if (talla == elArray.length) {

Figura nuevo[] = new Figura[2 \* talla];

System.arraycopy(elArray, 0, nuevo, 0, talla);

elArray = nuevo;

}

elArray[talla++] = c;

}

/\*\* Elimina la primera aparicion de "f" en un grupo de figuras

\* @param f Figura a eliminar

\* @return boolean true si se elimina con exito o false si "f" no esta en el grupo

\*/

public boolean eliminar(Figura f) {

boolean ok = false;

for (int i = 0; i < talla && !ok; i++)

if (elArray[i].equals(f)) {

ok = true;

for(int j = i; j < talla-1; j++)

elArray[j] = elArray[j+1];

talla--;

}

return ok;

}

/\*\* Devuelve la posicion de la primera aparicion de "f" en un grupo de figuras

\* @param f Figura a buscar

\* @return Posicion de la figura en el grupo (la primera figura ocupa la posicion 0) o -1 si "f" no esta en el grupo

\*/

public int indiceDe(Figura f) {

int pos = -1;

for (int i = 0; i < talla && pos == -1; i++)

if (elArray[i].equals(f)) pos = i;

return pos;

}

/\*\* Recupera la Figura del grupo que ocupa la posicion indicada

\* @param pos Posicion dentro del grupo (la primera figura ocupa la posicion 0)

\* @return Figura del grupo que ocupa la posicion pos, o null si la posicion no es correcta

\*/

public Figura recuperar(int pos) {

if (pos >= 0 && pos < talla) return elArray[pos];

return null;

}

/\*\* Consulta la talla del grupo

\* @return Numero de figuras en el grupo

\*/

public int talla() {

return talla;

}

/\*\* Devuelve la suma de las areas de todas las figuras del grupo

\* @return Suma de las areas de las figuras del grupo

\*/

public double area() {

double res = 0.0;

for (int i = 0; i < talla; i++)

res += elArray[i].area();

return res;

}

/\*\* Devuelve un array con las figuras del grupo ordenas de menor a mayor area

\* @return Array de tamaño talla() que contiene las figuras del grupo ordenados ascendentemente por area

\*/

public Figura[] ordenar() {

Figura aux, res[] = toArray();

for (int i = 1; i < talla; i++) {

for (int j = 0 ; j < talla - 1; j++) {

if (res[j].area() > res[j+1].area()) {

aux = res[j];

res[j] = res[j+1];

res[j+1] = aux;

}

}

}

return res;

}

/\*\* Devuelve un array con las figuras del grupo

\* @return Array de tamaño talla() que contiene todas las figuras del grupo

\*/

public Figura[] toArray() {

Figura nuevo[] = new Figura[talla];

System.arraycopy(elArray, 0, nuevo, 0, talla);

return nuevo;

}

/\*\* Devuelve una descipcion del grupo de figuras

\* @return Cadena de texto con la descripcion del grupo de figuras

\*/

public String toString(){

String res = "Grupo de " + talla + " figuras:\n";

if (talla > 0) {

for(int i = 0; i < talla; i++)

res += elArray[i].toString() + "\n";

} else res = "Grupo vac\u00edo\n";

return res;

}

}

package aplicaciones.figuras.gestionFiguras;

import aplicaciones.figuras.lasFiguras.\*;

/\*\* Clase TestJerarquiaFiguras: clase de prueba para validar la jararquia de figuras diseñada <br>

\* y el buen funcionamiento de la clase ArrayArrayGrupoDeFiguras. <br>

\* @author Profesores de la asignatura EDA, dsic-etsia, UPV

\* @since Oct 2020

\*/

public class TestJerarquiaFiguras

{

/\*\* Metodo principal \*/

public static void main() {

System.out.println("PROGRAMA DE PRUEBA PARA LA JERARQUIA DE FIGURAS");

System.out.println("-----------------------------------------------");

testConstructoresFiguras();

testEqualsFigura();

testLadoCuadrado();

ArrayGrupoDeFiguras gDF = new ArrayGrupoDeFiguras();

testInsertar(gDF);

testIndiceDe(gDF);

testEliminar(gDF);

testRecuperar(gDF);

testArea(gDF);

testToArray(gDF);

testOrdenar(gDF);

}

/\*\* Muestra por pantalla el resultado de la comprobacion \*/

private static void mostrarResultado(boolean res) {

if (res) System.out.println("\t... comprobaci\u00F3n correcta");

else System.out.println("\t... ERROR!!!");

}

/\*\* Comprueba los metodos constructores de la jerarquía de figuras \*/

private static void testConstructoresFiguras() {

boolean ok;

System.out.println("Comprobando los m\u00E9todos constructores de la jerarquía de figuras...");

Color red = Color.red, black = Color.black;

Circulo c1 = new Circulo();

Circulo c2 = new Circulo(red, 2.5);

Rectangulo r1 = new Rectangulo();

Rectangulo r2 = new Rectangulo(red, 1.5, 3.5);

Cuadrado u1 = new Cuadrado();

Cuadrado u2 = new Cuadrado(red, 2.5);

ok = c1.getTipo() != null && c2.getTipo() != null && r1.getTipo() != null &&

r2.getTipo() != null && u1.getTipo() != null && u2.getTipo() != null;

ok &= !c1.getTipo().equals("") && !c2.getTipo().equals("") && !r1.getTipo().equals("") &&

!r2.getTipo().equals("") && !u1.getTipo().equals("") && !u2.getTipo().equals("");

ok &= c1.getColor().equals(black) && r1.getColor().equals(black) && u1.getColor().equals(black) &&

c2.getColor().equals(red) && r2.getColor().equals(red) && u2.getColor().equals(red);

ok &= c1.getRadio()==10 && r1.getBase()==10 && r1.getAltura()==10 && u1.getLado()==10 &&

c2.getRadio()==2.5 && r2.getBase()==1.5 && r2.getAltura()==3.5 && u2.getLado()==2.5;

mostrarResultado(ok);

}

} // de la clase